Projektprotokoll – Monster Trading Cards Game (MTCG)

Projektname: Monster Trading Cards Game (MTCG)

Erstellungsdatum: 27.10.2024

GitHub Repo Link: https://github.com/Specifik/MTCG

1. Einleitung

Das Projekt „Monster Trading Cards Game (MTCG)“ ist eine serverseitige Anwendung, die Benutzerverwaltung und Spielsteuerung implementiert. Es basiert auf einer serviceorientierten Architektur mit einer klaren Trennung zwischen Service-, Controller- und Datenzugriffsschicht, um die Wartbarkeit und Erweiterbarkeit der Anwendung zu gewährleisten.

2. Designentscheidungen und Architektur

Die Anwendung verwendet eine REST-Architektur und nutzt das Repository- und Unit-of-Work-Muster, um eine konsistente und sichere Datenverwaltung zu gewährleisten. Hier sind einige der Hauptmodule und ihre spezifischen Rollen:

- **UserService:** Vermittelt zwischen HTTP-Anfragen und der Business-Logik der Benutzerverwaltung.
- **UserController:** Implementiert die Geschäftslogik für Benutzeraktionen wie Registrierung, Login, Profile abfragen und aktualisieren.
- **UserRepository:** Stellt eine Abstraktionsschicht für den Datenzugriff bereit und ermöglicht die Trennung von Geschäftslogik und Datenbankoperationen.
- **PackageController:** Verwalten von Kartenerstellungs- und Paketakquisitionsprozessen für die Benutzer, mit Authentifizierung für den Admin-Zugriff.
- **PackageRepository:** Bietet Datenbankoperationen zur Erstellung und Verwaltung von Kartensets und deren Besitz.

3. Klassenstruktur und -beschreibung

3.1 UserService

UserService implementiert das Service-Interface und verarbeitet HTTP-Anfragen, indem es Anfragen wie Benutzerregistrierungen oder -abfragen an den UserController weiterleitet.

- **Methode handleRequest(Request request):** Diese Methode verarbeitet GET- und POST-Anfragen und ruft die entsprechenden Controller-Methoden auf.

3.2 UserController

UserController ist verantwortlich für die Geschäftslogik in der Benutzerverwaltung und Authentifizierung. Er verwendet UserRepository für Datenbankoperationen und PasswordHasher für die Passwortverschlüsselung.

- **Methode addUser(Request request):** Registriert einen neuen Benutzer.
- **Methode loginUser(Request request):** Authentifiziert einen Benutzer und gibt ein Token zurück.
- **Methode getUser(String username, String token):** Ruft Benutzerdaten ab, wenn das Token gültig ist.
- **Methode updateUser(String username, Request request):** Aktualisiert Benutzerinformationen, mit Ausnahme des Passworts.

3.3 UserRepository

Die Klasse UserRepository stellt die zentrale Schnittstelle für den Datenbankzugriff in der Benutzerverwaltung dar. Sie kapselt die Datenbanklogik und ermöglicht eine klare Trennung zwischen der Geschäftslogik und dem Datenzugriff.

- **Methoden:**
    - registerUser: Registriert einen neuen Benutzer in der Datenbank.
    - findUserByToken: Sucht einen Benutzer anhand seines Tokens.
    - updateUserToken: Aktualisiert das Token eines Benutzers.
    - checkUser: Überprüft die Existenz eines Benutzers anhand des Tokens.

3.4 PackageController

PackageController verwaltet die Erstellung und Akquise von Kartensets. Nur Administratoren dürfen neue Pakete erstellen.

- **Methoden:**
    - **createPackage(Request request):** Erstellt ein neues Paket für den Benutzer, vorausgesetzt, er hat Administratorrechte.
    - **acquirePackage(Request request):** Ermöglicht es einem Benutzer, ein Paket zu erwerben, wenn er genügend Münzen hat.

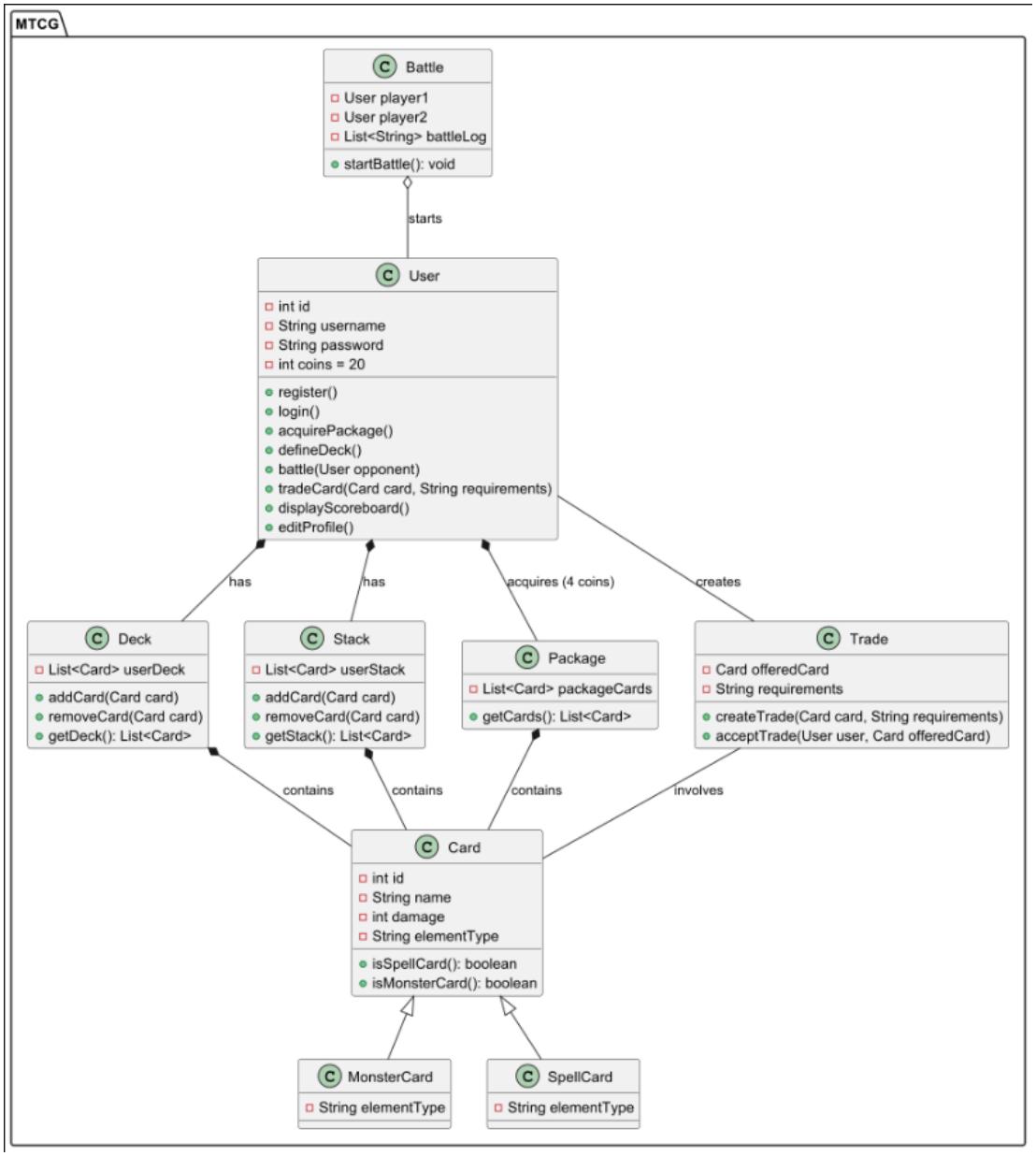
3.5 PackageRepository

Die Klasse PackageRepository bietet die Datenbankoperationen für die Erstellung und Verwaltung von Kartensets und deren Besitz.

- **Methoden:**
    - **createPackage:** Erstellt ein neues Kartenset und fügt es der Datenbank hinzu.
    - **acquirePackage:** Ermöglicht es einem Benutzer, ein Kartenset zu erwerben, das zuvor in der Datenbank gespeichert wurde.

4. Klassendiagramm (UML)

Projekt Klassendiagramm:



5. Erweiterungen und Änderungen

5.1 Unit Tests für UserController und PackageController

Im Rahmen der Entwicklung wurden mehrere Unit Tests für die Klassen **UserController** und **PackageController** erstellt, um die Funktionsweise der APIs zu validieren. Die Tests decken erfolgreiche und fehlgeschlagene Anfragen ab, einschließlich:

- Benutzerregistrierung
- Benutzerlogin
- Benutzerprofilabfrage
- Benutzerprofilaktualisierung
- Paket-Erstellung (nur durch Admin)
- Paket-Akquise (Verifizierung des Benutzers und seiner Münzen)

5.2 Verwendung von Mocking für Testfälle

In den Tests wurde **Mockito** verwendet, um die Interaktionen mit der Datenbank (via Repositorys) und andere externe Abhängigkeiten zu mocken. Auf diese Weise können die Tests ohne tatsächliche Datenbankinteraktionen ausgeführt werden.

5.3 Testbeispiele für Controller-Methoden:

- **testCreatePackage_Success:** Testet, ob ein Paket erfolgreich erstellt wird, wenn der Benutzer ein Administrator ist.
- **testCreatePackage_Fail:** Testet, ob ein Fehler 403 zurückgegeben wird, wenn der Benutzer nicht der Administrator ist.
- **testAcquirePackage_Success:** Testet, ob ein Benutzer ein Paket erfolgreich erwerben kann, wenn er genügend Münzen hat.
- **testAcquirePackage_NoAuthHeader:** Testet, ob eine 401-Antwort zurückgegeben wird, wenn der Authentifizierungstoken fehlt.
- **testGetUser_Fail_InvalidToken:** Testet, ob eine 401-Antwort zurückgegeben wird, wenn der Token ungültig ist.